Computational Geometry, Tsinghua University (Spring 2021)

画廊问题求解

游凯超 2020312661 软博 20
王兆伟 2020312653 软博 20
江俊广 2020213958 软硕 201

摘　要

基本问题：对二维画廊问题（给定一个简单多边形，选取若干个顶点安放摄像头，使得多边形内部的任何一个点都能被至少一个摄像头看到），利用 Fisk 的方法给出 $\lfloor \frac{n}{3} \rfloor$ 个哨兵的解决方案。

附加功能：按照计算几何课堂演示 demo 的质量要求，设计易用的图形界面，方便用户理解算法的实现过程。

目录

1 介绍

1975 年，Chvatal 发表的论文 [1] 表明画廊问题可以通过 $\lfloor \frac{n}{3} \rfloor$ 个哨兵来完整监视，且有时 $\lfloor \frac{n}{3} \rfloor$ 个哨兵是必要的。但是其证明非常冗长，且难以实现。三年后，Fisk 发表的论文 [2] 提供了一种极大简化了的证明，通过简单多边形的三角剖分及三染色优美地解决了这个问题。 Fisk 的证明虽然优美，但是依赖于简单多边形三角剖分的存在性及简单多边形无洞的事实。如何将一个简单多边形进行三角剖分？Chazelle [3] 在 1990 年证明，简单多边形可以在线性时间内得到三角剖分，但是这个算法太过复杂，几乎没有人去实现。更常见的做法 [4] 是采用两步：(1) 将简单多边形分解为若干个单调多边形；(2) 对每个单调多边形进行三角剖分。本次 project，我们将采用 [4] 中的两步方法，也即课上教的方法，对简单多边形进行三角剖分，具体包括

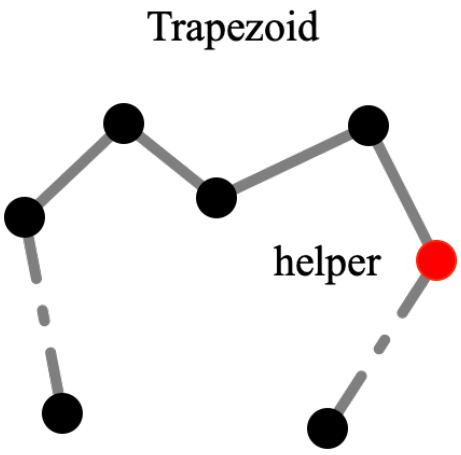
- 简单多边形分解成单调多边形；
- 单调多边形的三角剖分；
- 利用 Fisk 方法进行染色，生成 $\lfloor \frac{n}{3} \rfloor$ 个哨兵的监视方案；
- 三角剖分测例生成与正确性验证；
- 简单直观易用的图形界面（手动创建多边形、算法流程可视化）。

2 简单多边形分解成单调多边形

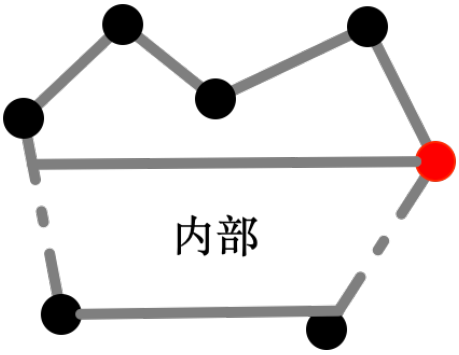
2.1 数据结构

四边形 (Trapezoid) 数据结构 Trapezoid（见图1a）由多个非封闭的单调多边形及两侧的边构成的。它包含一个 helper 顶点，以及顶点构成的边链。边链的两端有两个虚顶点（即还不属于 Trapezoid），根据这两个虚顶点及其附近的顶点，可以确定 Trapezoid 的两边延伸的方向，用于后续判断新的顶点是否在 Trapezoid 里面。图1b是 Trapezoid 的内部范围判定示意图（就是两侧虚边内所夹的平行线的范围）。

事件队列 扫描线算法需要扫描线状态与事件队列。在本问题中，事件队列就是按照 y 方向排好序的顶点。



(a) 典型的 Trapezoid



(b) Trapezoid 的内部范围判定示意图

基于平衡二叉搜索树的扫描线 (BBSTBasedSweepLine) 存储与当前扫描线相交的 Trapezoid 们。一般而言，扫描线状态如下，上面挂着一些 Trapezoid。随着扫描线的移动，需要动态地增加或者删除某个 Trapezoid。对于 N 边形而言，扫描线上挂着的 Trapezoid 个数为 $\mathcal{O}(N)$。

为了实现高效的查找以及增删，需要使用平衡二叉搜索树的结构维护 Trapezoid。Javascript 官方的库中并没有对平衡二叉搜索树的支持，因此我们参考了https://github.com/vadimg/js_bintrees 实现的二叉搜索树（BinarySearchTree）与红黑树（RBTree）。对于二叉搜索树而言，最重要的是定义清楚不同元素之间的序。观察到 Trapezoid T 与扫描线 SL 的右交点的 x 坐标 $\mathrm{INT}_{\mathrm{right}}(T, SL)$(图2) 满足反对称性和传递性，因此我们将其作为每个 Trapezoid 的 key。当扫描线向下移动时，每个 Trapezoid 的 key 是在动态变化的，但是 Trapezoid 之间的序并不会发生改变。具体实现时，不会保存每个结点的 key 值，而是在给定扫描线位置进行比较时才会计算 key 值。

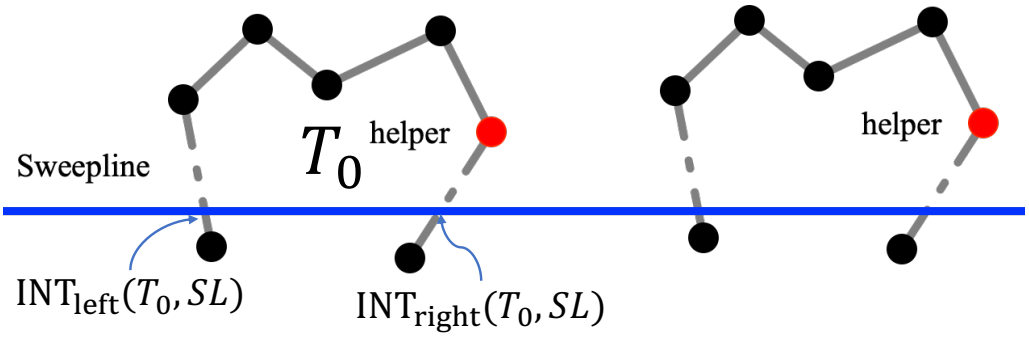


图 2: 不同 Trapezoid 的大小比较

在插入新的 Trapezoid 时，只涉及到不同 Trapezoid 之间的比较，即比较 $\mathrm{INT}_{\mathrm{right}}(T, SL)$ 大小。插入后树可能不再满足红黑树的特性，因此需要通过旋转和重新着色等方法来修正该树，使之重新成为一颗红黑树。该过程的复杂度是 $\mathcal{O}(\log(N))$。删除旧的 Trapezoid 的过程类似，复杂度也是 $\mathcal{O}(\log(N))$。

在查找时，我们需要用一个点 P 去和 Trapezoid 进行比较，由于这个点一定在扫描线上，因此我们直接将该点 P 的 x 坐标 x_P 定义为 key 值。我们在树中二分查找找到 key 值大于等于 x_P 的首个结点 T_0，然后从 T_0 开始顺序查找符合条件的 Trapezoid。由于序小于 T_0 的结点对应的 Trapezoid 与扫描线的右交点 x 坐标均小于 x_P，因此不会包含点 P，故而不会出现遗漏的问题。查找的终止条件是当前的 Trapezoid T 与扫描线 SL 的左交点的 x 坐标 $\mathrm{INT}_{\mathrm{left}}(T, SL)$ 大于 x_P。由于包含点 P 的 Trapezoid 的个数不会超过 2 个，同时红黑树的树高不会超过 $\mathcal{O}(\log(N))$，因此在 BBSTBasedSweepLine 中查找某个点的复杂度是 $\mathcal{O}(\log(N))$。

扫描线移动次数为 $\mathcal{O}(N)$，因此扫描过程总的时间复杂度为 $\mathcal{O}(N\log(N))$。

2.2 算法流程

从上到下第一遍扫描

1. 将所有点按照 y 坐标排序，从上到下逐个处理
2. 根据当前点与扫描线上的四边形关系，决定如何处理。

将所有点按照 y 坐标排序，从上到下逐个处理根据当前点与扫描线上的四边形关系，决定如何处理。处理扫描线状态时，注意更新边链和 helper 顶点。共有以下六种情况：

Case 1: insert Trapezoid 顶点不在任何一个 Trapezoid 里面，则顶点与它左右的两个点构成一个新的 Trapezoid，插入 sweepline 中。

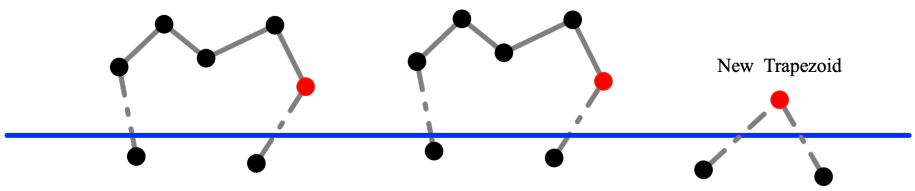


图 3: Trapezoid 插入的情况

Case 2: output Trapezoid 顶点在当前 Trapezoid 里面，且这个 Trapezoid 的左右虚顶点重合，表示这个 Trapezoid 已经封闭，可以输出了。

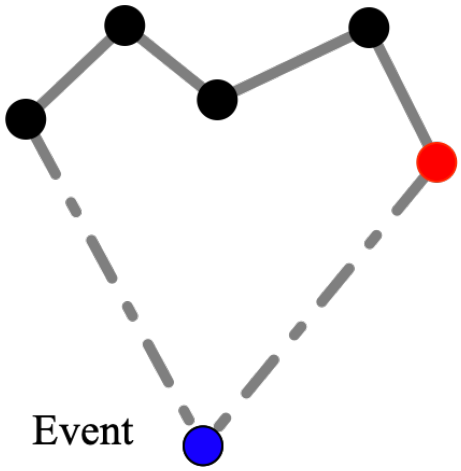


图 4: 输出 Trapezoid 的情况

Case 3/4: Trapezoid left/right update 顶点是左边/右边的虚顶点，则按照课件的步骤更新即可。

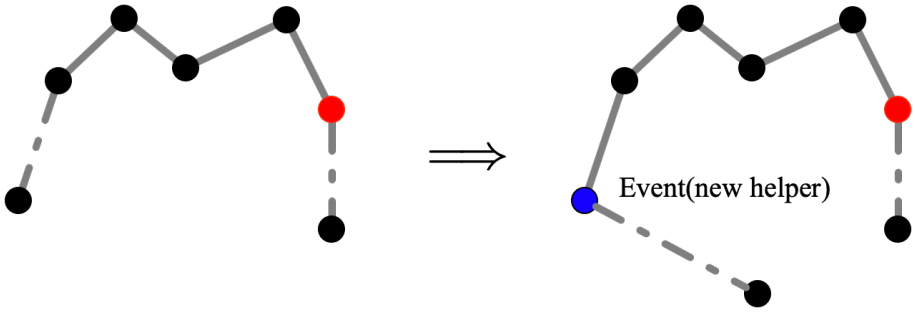


图 5: Trapezoid 更新 helper 顶点的情况

Case 5: Trapezoid merge 顶点是两个 Trapezoid 的连接点，则合并两个 Trapezoid 即可。

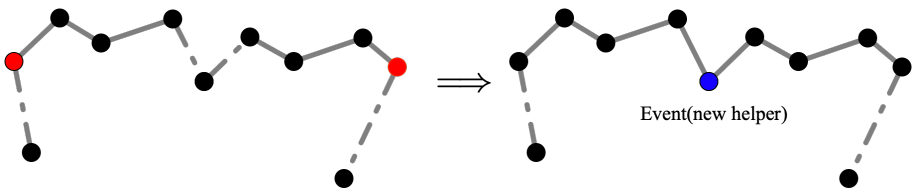


图 6: Trapezoid 合并的情况

Case 6: Trapezoid split 顶点在某一个 Trapezoid 里面，且不是左右虚顶点。则需要分成两个 Trapezoid。

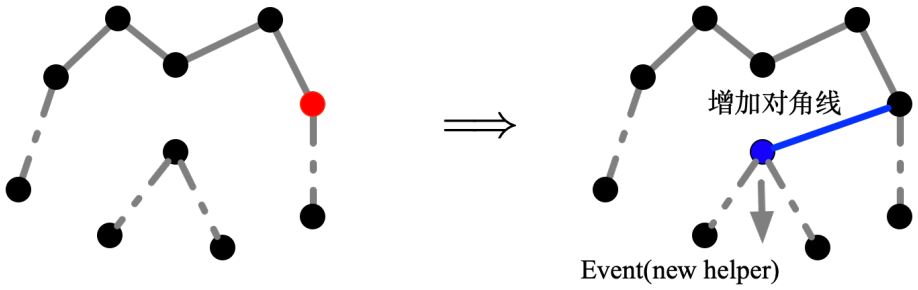


图 7: Trapezoid 分裂（增加对角线）的情况

从下到上第二遍扫描 按照如上操作，则简单多边形可以被分解为若干个 Trapezoid。每一个 Trapezoid 不保证是单调多边形，但是对 Trapezoid 再来一次从下往上的 Trapezoid 分解，则可完成单调多边形分解。图8是一个典型例子。

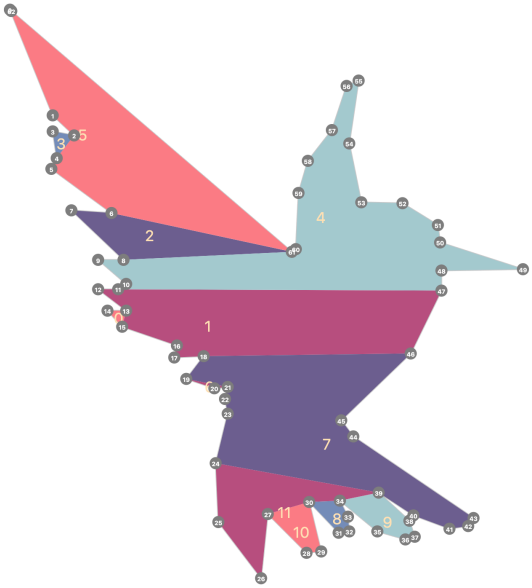


图 8: 单调多边形分解的典型例子

3 单调多边形三角剖分

给定一个边数为 M，竖直方向单调的单调多边形（包含左右两个竖直方向单调的链），我们采用扫描线算法对其进行三角剖分。

3.1 数据结构

事件队列 扫描线算法需要扫描线状态与事件队列。在本问题中，事件队列就是按照 y 方向排好序的顶点。在实际应用中，两个顶点 y 坐标相同的情况非常常见，若这两个顶点属于同一个链，那么它们在事件队列中的次序应该和链中的次序保持一致，因此直接对所有结点使用普通的排序算法既不高效也不保序。合理的方法是，给定一个结点序列（多边形结点的逆时针排序），找到 y 坐标最小和最大的两个结点，从而得到左右两条 y 单调链，然后对这两条链按照 y 坐标进行归并得到事件队列。这样能够保证事件队列中的次序和链中的次序的一致性，总的时间复杂度为 $\mathcal{O}(M)$。

3.2 算法流程

维护一个结点栈（Vertex Stack）S，保存扫描线已经经过但是尚未处理的结点。将栈首元素记作 t, 栈中第二个元素记作 s，扫描线当前经过的元素记作 c。

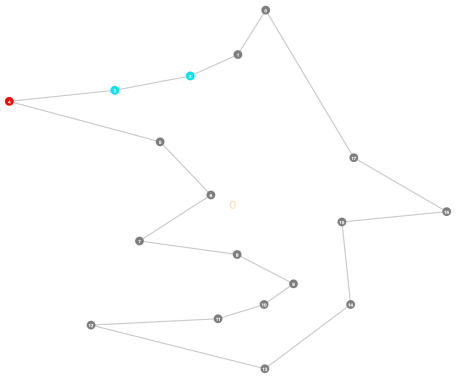
核心想法是当 s, t, c 构成了一个凸起的三角形，我们可以像割耳朵一样将这个三角形割掉，因此我们需要判断当前的三角形是否凸起。对于左链而言，就是对 toLeft 测试为 true，对于右链而言 toRight 测试为 true。我们将相反的情况，即 s, t, c 构成凹陷的三角形或者共线称作 t 为 reflex。以下，我们对 c, t, s 之间的不同的相对位置，分别进行处理。

Case A1: Same Side + Reflex 当 c 和 t 处于同一条链条上，并且 t 是 reflex 的 (图9a)，此时不能割下任何三角形，因此我们将 c 压入栈中。

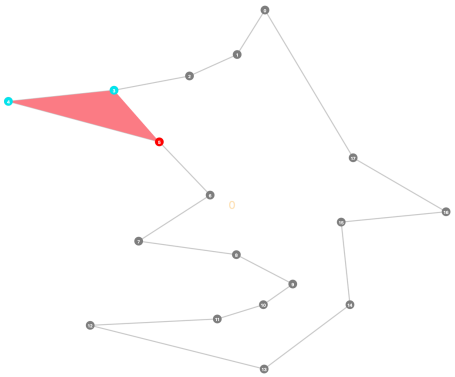
Case A2: Same Side + Non-Reflex 当 c 和 t 处于同一条链条上，并且 t 不是 reflex 的 (图9b)，则可以割下三角形 s, t, c。若构成新的凸三角形, 则需要继续割三角形。当 t 再次 reflex 或者栈中元素只有一个时，停止循环。完整算法见算法1。

Case B: Opposite Side 当 c 和 t 处于不同的链条上时 (图10)，此时 c 与栈 S 中的任意两个相邻结点都构成了三角形，因此可以将 S 中所有元素弹出。此后 t 会成为新的栈底元素，c 会成为新的栈顶元素。完整算法见算法2。

由于每个结点至多被压入栈中两次，至多产生 $M-2$ 个三角形，因此总共的时间复杂度是 $\mathcal{O}(M)$。



(a) Case A1: Same Side + Reflex。



(b) Case A2: Same Side + Non-Reflex。

图 9: 多边形的三角剖分。红色:c, 蓝色:s, t

算法 1 单调多边形分解。Case A2: Same Side + Non-Reflex

repeat

$s = S$.secondary_top();

ChopOffTriangle(s, t, c);

$t = S$.pop();

until (t is reflex) **or** ($|S| == 1$);

S.push(c);

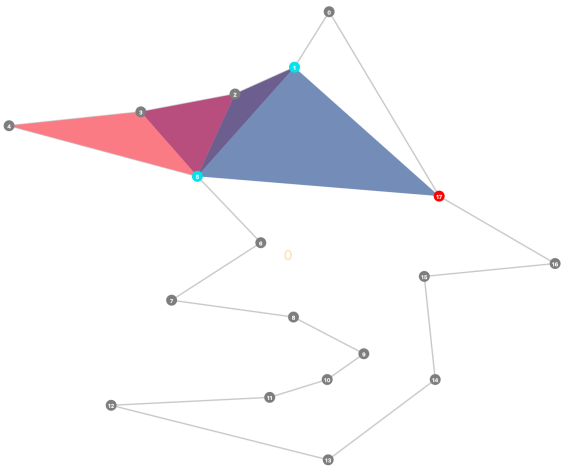


图 10: Case B: Opposite Side。红色:c, 蓝色:s, t

算法 2 单调多边形分解。Case B: Opposite Side

$top = t$;
repeat
  ChopOffTriangle(s, t, c);
  $t = S$.pop();
until $(|S| == 1)$;
S.pop(); S.push(top); S.push(c);

4 Fisk 染色

给定 N 边形的三角剖分集合，我们首先将其用 DCEL 进行存储，然后将其转化为对偶图。对于不带洞的多边形而言，其三角剖分的对偶图是一棵树。最后采用深度优先遍历算法遍历这棵树，得到最终的 Fisk 染色。

4.1 数据结构

Doubly connected edge list (DCEL) DCEL 是基于边来存储平面图划分的方法, DCEL 会记录所有的结点, 边以及面。其中, 每条边会使用两条半边 (HalfEdge) 进行表达。

- 对于每个结点 v，我们会存储所有从 v 出发的半边 edges(v)。
- 对于每条半边 e, 我们会存储它的起点 src(e)，终点 tgt(e), 方向相反的另外一条半边 twin(e) 和相邻的面 inc(e)。
- 对于每个面 f, 我们会存储它边界上的所有边 edges(f)

由于所有结点的度数和为边数的两倍，所有面的边数和不超过边数的两倍，因此上述 DCEL 的存储成本 $\mathcal{O}(N)$。

在 Fisk 染色问题中，我们需要支持的功能是给定任意一个面，返回所有与之相邻的面。

4.2 算法流程

首先将三角剖分转成对应的 DCEL（算法3）。

然后在 DCEL 结构上构建三角剖分 T 的对偶图 (算法4)。该过程的时间复杂度只取决于总边数，因此为 $\mathcal{O}(N)$。

算法 3 DCEL 的构建

输入: N 边形 P, 对应的三角剖分 T

输出: DECL D

for all 顶点 $p \in P$ **do**

D.createVertex(p);

end for

for all 三角形 $t \in T$ **do**

令 va, vb, vc 是三角形 t 的三个顶点;

$eab = D$.createEdge(va, vb);

$ebc = D$.createEdge(vb, vc);

$eca = D$.createEdge(vc, va);

D.createFace(eab, ebc, eca);

end for

算法 4 对偶图的构建

输入: DECL D

输出: 对偶图 DUAL(D)

for all 面 $f \in faces(D)$ **do**

for all 邻边 $e \in edges(f)$ **do**

$f' = \text{twin}(e)$

if $f' \neq$ NULL **then**

连接顶点 DUAL(f) 和 DUAL(f')

end if

end for

end for

对偶图 D 中每个顶点对应三角剖分 T 的一个三角面片，顶点之间的连边对应三角面片之间相邻。在对三角剖分进行 $Fisk$ 染色时，只需先选一张三角面片染三种颜色，再使用深度优先遍历算法 BFS 逐个对相邻的三角面片的顶点染色即可 (算法5)。由于每张三角面片仅访问一次，因此该过程的时间复杂度取决于三角面片的个数，因此为 $\mathcal{O}(N)$。

算法 5 $Fisk$ 染色

输入: 对偶图 DUAL(D), 三角剖分 T

输出: 染色后的三角剖分 T

```
令 s 为一个空的队列, c 为 DUAL(D) 的任意一个顶点;
s.push(c)
while !s.empty do
    c=s.top;
    s.pop();
    对 c 对应的三角面片进行染色;
    for c ∈ c.neighbors do
        if c 还未被染色 then
            s.push(c)
        end if
    end for
end while
```

5 高效的三角剖分正确性性检查

由于任意多边形的三角剖分不唯一，因此无法通过和人工三角剖分结果比对的方式验证准确性。我们希望能找到一个尽可能高效的正确性检查方式，并尽可能将所有错误的剖分都检查出来，为此我们首先给出下述定理及证明。

定理 5.1 (三角剖分充要条件). *给定简单 N 边形 P, 三角形集合 T 是 P 的三角剖分的充要条件是*

$$E_1 = \{P\text{的所有边}\}; |E_2| = N - 3; |E_k| = 0, k > 2 \tag{1}$$

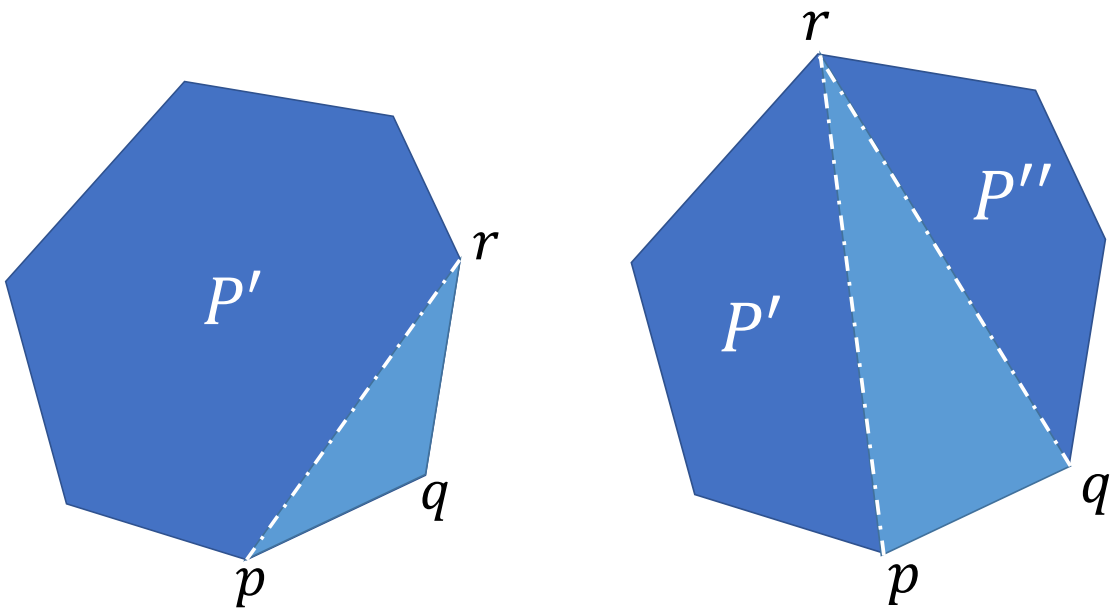
其中 E_k 的定义如下。

定义 1 (E_k 的定义). 给定三角形集合 T,E 是所有三角形的边构成的有重集合，即

$$E = \{e|t \in T, e \in t\}, \tag{2}$$

E_k 是 E 中出现 k 次的边构成的集合，即

$$E_k = \{e|\text{count}(e) = k, e \in E\} \tag{3}$$



(a) 情况 1　　(b) 情况 2

图 11: 当 $N = K$ 时的多边形情况。

定理5.1的证明. 必要性易证。以下采用数学归纳法证明充分性。当 $N = 3$ 时，充分性是显然的。假设当 $N < K$ 时，充分性满足。则当 $N = K$ 时，

情况 1: P 上存在相邻的三点 p, q, r 构成 T 中的一个三角形 Δ_{pqr}（见图11a)。令 P' 是多边形去除 Δ_{pqr} 后剩余的部分，$T' = T - \{\Delta_{pqr}\}$。不难验证下述式子成立。

$$E'_1 = \{P'\text{的所有边}\}; |E'_2| = N - 4; |E'_k| = 0, k > 2 \tag{4}$$

由归纳假设知，E' 是 P' 的三角剖分, 因此 E 也是 P 的三角剖分。

情况 2: P 上不存在相邻的三点 p, q, r 构成 T 中的一个三角形 Δ_{pqr}。那么不妨取 P 上相邻的两点 p, q，它参与构成 T 中的一个三角形 Δ_{pqr}，并且 r 与 p, q 都

不相邻，此时 P 被分成了 P', P'' 和 Δ_{pqr} 三部分。我们用 E'_k 表示 E_k 中只包含 P' 中点的边所构成的子集，用 E''_k 表示 E_k 中只包含 P'' 中点的边所构成的子集。在 P' 中，rp 之外的边都参与构成 P' 中不同的三角形（否则违反情况 2 的假设），故 T' 中至少有 $|P'|-1$ 个三角形，故

$$|E'_1| + 2|E'_2| \geq 3(|P'| - 1), \tag{5}$$

同理

$$|E''_1| + 2|E''_2| \geq 3(|P''| - 1), \tag{6}$$

结合 $|E'_1| + |E''_1| \leq |E_1| + 1 = N + 1$ 与 $|P'| + |P''| \geq |P| + 1 = N + 1$ 得，

$$|E'_2| + |E''_2| \geq \frac{1}{2}[3(|P'| - 1) + 3(|P''| - 1) - (|E'_1| + |E''_1|)] = N - 2. \tag{7}$$

由于 $|E_2| \geq |E'_2| + |E''_2| + 2 \geq N$, 与公式1矛盾！故情况 2 不存在。

综上，由数学归纳法知，定理5.1成立。 □

在得到定理5.1后，我们只需要验证1即可。由于实际的三角剖分算法获得的 T 不可能引入 P 之外的结点，因此实际测试时，只需要验证

$$|E_1| = N; |E_2| = N - 3; |E_k| = 0, k > 2 \tag{8}$$

即可。具体验证过程见算法6。期望意义下，Hash 的插入复杂度均为 $\mathcal{O}(1)$。因此算法6总的时间复杂度不超过总边数，为 $\mathcal{O}(N)$。

算法 6 三角剖分正确性验证

输入: N 边形 P, 对应的三角剖分 T

输出: 三角剖分正确性

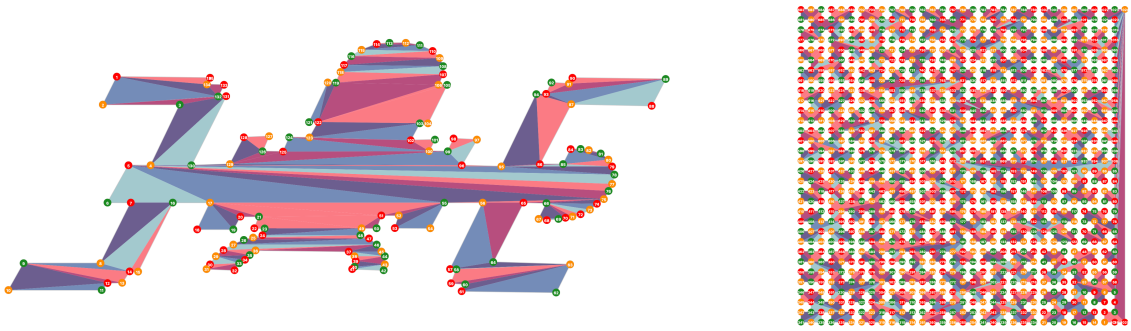
```
counter = new Counter(); /* 采用 Hash 结构实现计数器 */
for all 三角形 t ∈ T do
  for all 边 e ∈ t do
    counter.add(e);
  end for
end for
/* 统计各个出现次数对应的边的集合 */ /* 采用 Hash 结构实现集合 */
E_1 = new Set(), E_2 = new Set(), E_k = new Set();
for all 边 e, 出现次数 c ∈ counter.items() do
  if c == 1 then
    E_1.add(e); /* 出现次数等于 1 的边的集合 */
  else if c == 2 then
    E_2.add(e); /* 出现次数等于 2 的边的集合 */
  else
    E_k.add(e); /* 出现次数大于 2 的边的集合 */
  end if
end for
if E_1.size()== N and E_2.size()== N − 3 and E_k.size()== 0 then
  return true;
else
  return false;
end if
```

6 鲁棒性

本文的目标是画廊问题，画廊问题的输入一般是室内设计图，因此点与点之间的最小距离能够得到保障，重合点可以看做是一个点。因此，我们的算法只处理点的最小距离比较大（大于 10^{-3} 像素距离）的情况。本节所谓的鲁棒性指的是算法对平行、共线等计算几何算法的常见边界条件的鲁棒性。

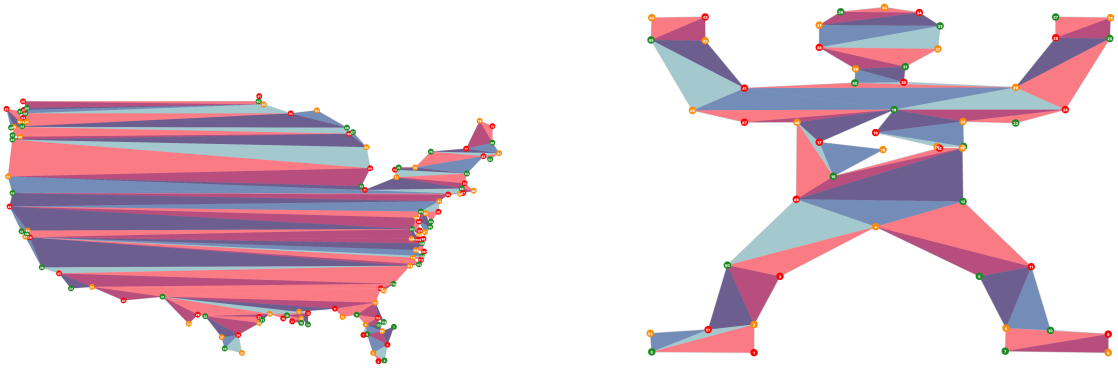
6.1 正确测例

室内设计图、迷宫 这种测例的特点是形状比较规则（图12），存在多条平行于 x 方向或者 y 方向的边。由于标准的单调多边形分解算法要求不同点的 y 坐标不同，因此我们采用了扰动算法，将输入做微小的扰动再进行处理，使得程序能够处理这些测例。



(a) 清华主楼　　(b) 迷宫

图 12: 室内设计图、迷宫。

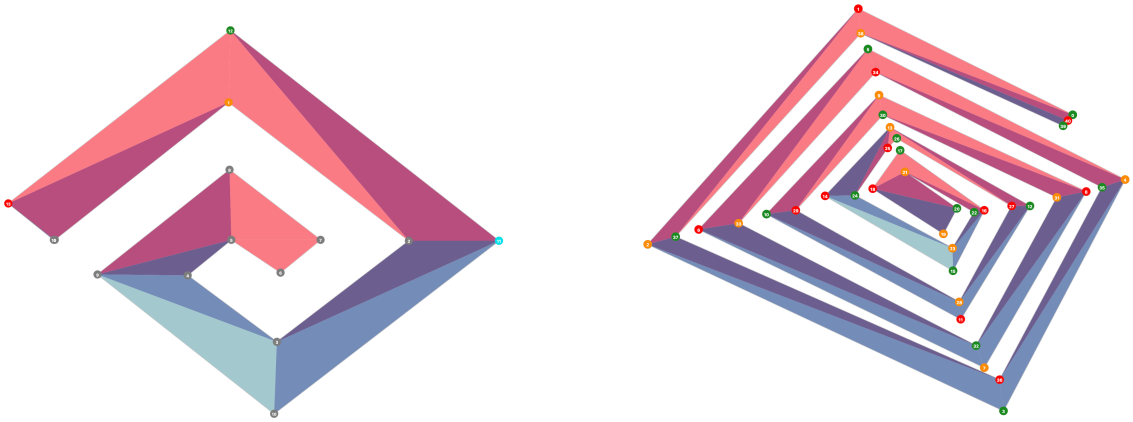


(a) 美国的国土边界　　(b) 人体。

图 13: 物体外形、地形图。

物体外形、地形图　这种测例的特点是外表面非常复杂（图13），例如人体、鸟、马、国土边界等。其中人体测例（看上去有洞，但是实际上没有洞）展示了，如果输入的多边形有洞，可以通过连接内洞和多边形外表面，将其转化为不带洞的多边形，然后调用简单多边形的剖分算法进行三角剖分。

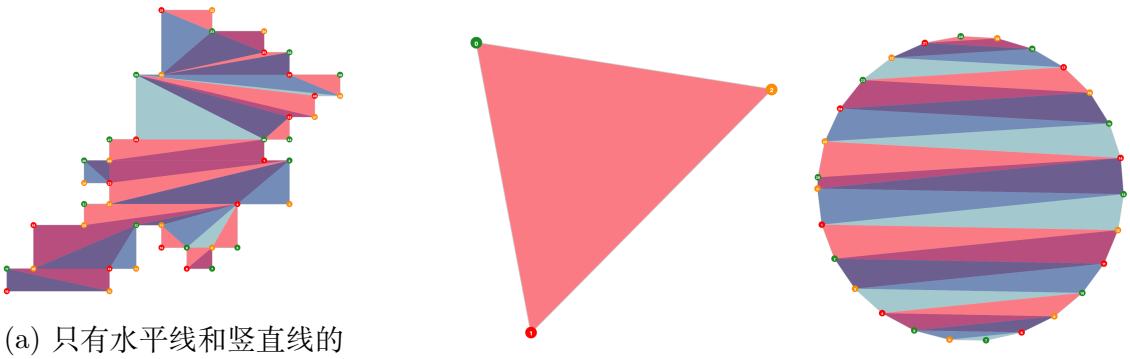
多 sinuosity　Sinuosity 描述了多边形的蜿蜒度 [5]。本文提供了蜿蜒度较高的测例（图14）。



(a) 3-Sinuosity　　(b) More sinuosity

图 14: 多 Sinuosity。

退化的测例　退化情况的测例包括三角形，圆等（图15）。



(a) 只有水平线和竖直线的测例（有较多共线点）。　(b) 三角形　(c) 圆

图 15: 退化的测例。

大规模测例　当测例的规模较大时，已经很难通过可视化去判断结果的正确性了（图16），因此在性能测试部分只给出定量的结果。

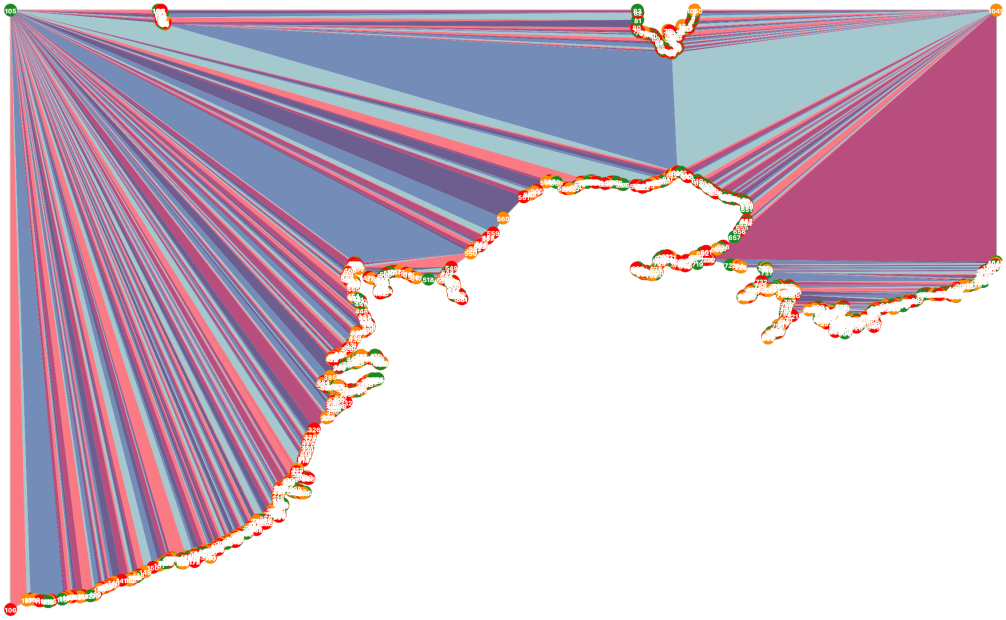
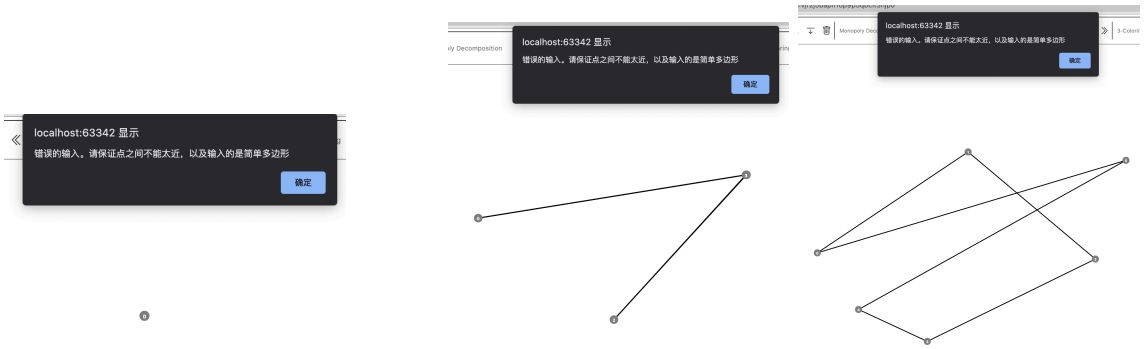


图 16: 大规模测例 ($N = 1051$)。

6.2 错误测例

当输入的不是简单多边形时，例如单点、点重合、非平面图等情况时，系统会给出错误提示（图17）。



(a) 只有一个点的测例。　(b) 点重合的测例　(c) 非平面图的测例

图 17: 错误的输入。

7 性能评估

我们记录了当多边形的点数 N 从 10^3 增加到 10^6 过程中，程序运行总时间的变化。对于 $\mathcal{O}(N^p)$ 复杂度的算法，时间和规模取对数后，拟合的斜率就是 p 值。

本题中我们实现的算法是 $\mathcal{O}(N\log(N))$ 的，斜率应该在 1 到 2 之间。图18 验证了我们的猜想。

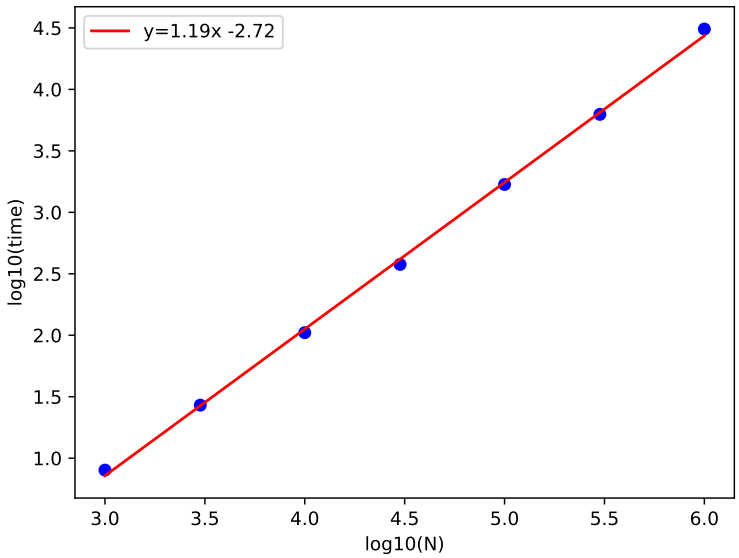


图 18: 运行总时间 (ms) 随着数据规模的变化。横纵坐标均为取对数后的值。

8 总结

我们小组基于 JavaScript 实现了画廊问题求解，并基于 HTML5 制作了美观大方的界面，只需要有浏览器就可以运行，其效果足以供课堂教学使用，且有足够的效率和鲁棒性。

9 分工

- 游凯超：简单多边形分解成单调多边形。
- 王兆伟：图形界面（包括算法流程可视化，多边形的创建、上传与下载），Fisk 染色。
- 江俊广：单调多边形的三角剖分，基于红黑树的扫描线，DCEL，剖分正确性检查。

参考文献

[1] Chvatal Vasek, A combinatorial theorem in plane geometry, Journal of Combinatorial Theory Series B, 1975.

[2] Fisk Steve, A short proof of Chvatal's watchman theorem, Journal of Combinatorial Theory Series B, 1978.

[3] Chazelle, Triangulating a simple polygon in linear time, FOCS, 1990

[4] Garey Michael, Johnson David, Preparata Franco, Tarjan Robert, Triangulating a simple polygon, Information Processing Letters, 1978

[5] Bernard Chazelle and Janet Incerpi, Triangulation and Shape-Complexity, ACM Transactions on Graphics,1984